# Scalable, parallel computation of the translation operator in three dimensions

Bart Michiels [1], Ignace Bogaert [1], Jan Fostier [1], and Daniël De Zutter [1]

[1] *Ghent University, Ghent, B-9050, Belgium*
jan.fostier@intec.ugent.be

***Abstract*— We propose a novel algorithm for the parallel, distributed-memory computation of the translation operator in the three-dimensional Multilevel Fast Multipole Algorithm (MLFMA). Sequential algorithms can compute the translation operator with $L$ multipoles and $\mathcal{O}(L^2)$ sampling points in $\mathcal{O}(L^2)$ time. State-of-the-art hierarchical parallelization schemes of the MLFMA rely on the distribution of radiation patterns and associated translation operators among $P = \mathcal{O}(L^2)$ parallel processes, necessitating the development of distributed-memory algorithms for the computation of the translation operator. Whereas a baseline parallel algorithm computes this translation operator in $\mathcal{O}(L)$ time, we propose an algorithm that achieves this in only $\mathcal{O}(\log L)$ time. For large translation operators and a high number of parallel processes, our algorithm proves to be roughly ten times faster than the baseline algorithm.**

## I. Introduction

The multilevel fast multipole algorithm (MLFMA) allows for the evaluation of the fields in $N$ points due to $N$ electromagnetic sources in $\mathcal{O}(N \log N)$ time. We assume the reader is familiar with the MLFMA and its terminology. For a good introduction, we refer to [1]. Within the MLFMA, sources and evaluation points are hierarchically organized in an octree of boxes and interactions between these boxes are evaluated using so-called translation operators. The translation operator with $L$ multipoles is given by:

$$T(\vec{k}, \vec{R}_T) = \sum_{l=0}^{L} (-j)^l (2l+1) h_l^{(2)}(kR_T) P_l(\cos\theta_T) \qquad (1)$$

with $\cos\theta_T = \vec{1}_k \cdot \vec{1}_{R_T}$, $\vec{k} = k\vec{1}_k$ a vector representing the angular direction in which the translation operator is to be evaluated, $k$ the wavenumber, $\vec{R}_T = R_T \vec{1}_{R_T}$ the translation direction connecting the centres of the two interacting boxes and $P_l(.)$ and $h_l^{(2)}(.)$ the Legendre polynomial and spherical Hankel function of the second kind and order $l$ respectively. Given a fixed $k$ and $\vec{R}_T$, the translation operator $T$ is a one-dimensional function of $\theta_T$.

As radiation patterns with $L$ multipoles are sampled in $\mathcal{O}(L^2)$ directions, a direct computation using equation (1) requires $\mathcal{O}(L^3)$ time, as there are $L+1$ terms to evaluate per direction. This method is referred to as the 'direct method' (DM). In [2], a two-step approach was proposed to compute the translation operator in only $O(L^2)$ time: in a first step, the band-limited function $T(\theta_T)$ is evaluated in $O(L)$ equidistant interpolation points in the $[0 \dots \pi]$ interval using (1). In a second step, the $O(L^2)$ points that make up the translation operator are computed through local interpolation. This method is referred to as the 'interpolation method' (IM).

In this work, we present a novel algorithm for the parallel, distributed-memory construction of the translation operator in three dimensions. This work is closely affiliated to recent developments of parallel, distributed-memory MLFMA algorithms. Advanced implementations, based on a hierarchical distribution of radiation pattern samples, rely on the distribution of these samples among an increasing number of $P = 1, 4, 16, \dots, 4^k$ parallel processes for every higher level in the MLFMA [3], [4]. Formally, radiation patterns are distributed among $P = \mathcal{O}(L^2)$ parallel processes. During the translation phase, each process operates only on the subset of sampling points that is locally contained in memory. Consequently, each process requires only a corresponding subset of the translation operator sampling points. These are typically precomputed during the setup stage of the MLFMA. To the best of our knowledge, algorithms for the distributed-memory computation of translation operators have not been studied to date.

## II. Parallel Algorithm

In what follows, we always consider the case where a translation operator with $L$ multipoles, sampled in $\mathcal{O}(L^2)$ directions, is computed using $P = \mathcal{O}(L^2)$ processes. This means that each process needs to compute only $\mathcal{O}(1)$ sampling points. A baseline, communication-free parallel algorithm, referred to as the 'parallel direct method' (PDM), simply involves each process evaluating their local points using equation (1) directly. This however, gives rise to a computational complexity of $\mathcal{O}(L)$, as there are $L+1$ terms to evaluate in equation (1). This is sub-optimal.

We present an algorithm that is based on the parallelization of the IM and that is further referred to as the 'parallel interpolation method' (PIM). To parallelize the first step, the $P = O(L^2)$ processes involved in the computation of the translation operator are divided in $\sqrt{P}$ groups, each group consisting of $\sqrt{P}$ processes. The $\mathcal{O}(L)$ interpolation points in the $[0 \dots \pi]$ interval are then partitioned among the $\sqrt{P} = \mathcal{O}(L)$ process groups. Hence, each group of processes is responsible for the computation of $\mathcal{O}(1)$ interpolation points. The computations within a group are further parallelized as follows: each process within a group computes and sums only a subset of the $L+1$ terms from equation (1). Because

there are $\sqrt{P} = \mathcal{O}(L)$ processes per group, each process computes only $\mathcal{O}(1)$ terms for each of the $\mathcal{O}(1)$ interpolation points in its group. These computations take $\mathcal{O}(1)$ time and no computations are duplicated between processes. It is important to remark that we hereby assume that Legendre polynomials and spherical Hankel functions can effectively be evaluated in $\mathcal{O}(1)$ time, regardless of the order $l = 0 \ldots L$. For the spherical Hankel functions, such libraries already exist (e.g., Amos). For the Legendre polynomials, such a method has recently been developed in [5].

Next, these partial results are summed over the parallel processes in each group, such that the resulting sum (corresponding to the value of an interpolation point) is present in each process of the group. This parallel summation is well-known in parallel computing and is often referred to as an all-reduce operation. It requires $\mathcal{O}(\log L)$ time, which is also the dominant complexity term during the first step of this algorithm.

The parallelization of the second step of the IM is conceptually straightforward: each process computes the required $\mathcal{O}(1)$ sampling points of its local partition of the translation operator through interpolation. For this, certain interpolation points computed during the first step are required. As a single sample point requires only a constant number ($\mathcal{O}(1)$) of interpolation points, the total number of interpolation points required by a process is also $\mathcal{O}(1)$. However, these required interpolation points are not necessarily the ones that were locally computed during the first step. Therefore, a reshuffling of interpolation points is required between both steps. This communication phase is non-trivial, and only after quite lengthy calculations, one can prove that the volume of communication per process is bounded by $\mathcal{O}(\log L)$ [6]. Assuming all computations and communications by the different process can be concurrently executed, the PIM algorithm computes the translation operator in $\mathcal{O}(\log L)$ time, using $P = \mathcal{O}(L^2)$ processes.

## III. Results

Table I presents the calculation time for a single translation operator using the sequential DM and IM methods, and the parallel PDM and PIM methods. This numerical data was obtained using a cluster consisting of 256 machines each containing two 8-core Intel Xeon E5-2670 processors (4096 CPU cores in total). The machines were connected using an Infiniband network. The timings are presented for different levels of the MLFMA, and hence different $L$. The number of processes for the parallel methods is taken proportional to $\mathcal{O}(L^2)$, to correspond to the situation encountered in the hierarchical MLFMA. From Table I, one can observe the $\mathcal{O}(L^3)$, $\mathcal{O}(L^2)$, $\mathcal{O}(L)$ and $\mathcal{O}(\log L)$ complexities for the DM, IM, PDM and PIM methods respectively. By comparing the PDM and PIM method, it can be observed that the PIM outperforms the PMD by a factor of roughly ten for the larger translation operators. This is a manifestation of its lower time complexity. In a realistic MLFMA simulation, during the setup stage, a few hundreds of translation operators need to be computed for every level of the tree. As a result, the PIM method reduces the total runtime for the computation of the translation operators from hours to only minutes, thus removing a bottleneck that is becoming apparent for large-scale MLFMA simulations.

TABLE I

Runtime (seconds) to compute a single translation operator for an increasing number of multipoles (target precision $\epsilon = 10^{-6}$), in case of the direct method (DM), the interpolation method (IM), the parallel direct method (PDM) and the proposed parallel interpolation method (PIM).

| $L$ | DM $O(L^3)$ | IM $O(L^2)$ | $P$ | PDM $O(L)$ | PIM $O(\log L)$ |
|---|---|---|---|---|---|
| 170 | 0.90 | 0.53 | 4 | 0.24 | 0.143 |
| 316 | 5.62 | 1.92 | 16 | 0.38 | 0.138 |
| 604 | 38.81 | 6.82 | 64 | 0.67 | 0.151 |
| 1171 | 280.1 | 26.5 | 256 | 1.22 | 0.170 |
| 2295 | 2100 | 98.5 | 1024 | 2.38 | 0.251 |
| 4532 | 16132 | 404.3 | 4096 | 4.67 | 0.575 |

## IV. Conclusions

A novel parallel, distributed-memory algorithm is presented that computes the translation operator in the three-dimensional MLFMA in $\mathcal{O}(\log L)$ time using $P = \mathcal{O}(L^2)$ parallel processes. The algorithm has been implemented and tested up to 4096 parallel processes. For realistic but large values of $L$ and $P$, the algorithm is roughly ten times faster than a baseline parallel algorithm. The algorithm reduces the computation of the translation operators during the setup stage of the MLFMA from hours to only minutes, for large-scale simulations.

## Acknowledgment

The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by Ghent University, the Hercules Foundation and the Flemish Government – department EWI. The work of B. Michiels was supported by a doctoral grant from the Special Research Fund (BOF) at Ghent University. The work of I. Bogaert was supported by a post-doctoral grant from Research Foundation-Flanders (FWO-Vlaanderen).

## References

[1] W.C. Chew, J. Lin, E. Michielssen and J. Song, *Fast and Efficient algorithms in Computational Electromagnetics*. Boston: Artech House, 2001.

[2] J. Song, and W.C. Chew, "Interpolation of Translation Matrix in MLFMA," *Microwave and Optical Technology Letters*, vol. 30, no. 2, pp. 109–114, July 2001.

[3] Ö. Ergül, and L. Gürel, "Hierarchical Parallelisation Strategy for Multilevel Fast Multipole Algorithm in Computational Electromagnetics," *Electronics Letters*, vol. 44, no. 1, pp. 3–4, Jan. 2008.

[4] J. Fostier, and F. Olyslager, "Provably Scalable Parallel Multilevel Fast Multipole Algorithm," *Electronics Letters*, vol. 44, no. 19, pp. 1111–1112, Sept. 2008.

[5] I. Bogaert, B. Michiels, and J. Fostier, "O(1) Computation of Legendre Polynomials and Gauss-Legendre Nodes and Weights for Parallel Computing," *SIAM Journal on Scientific Computing*, vol. 34, no. 3, pp. C83–C101, May. 2012.

[6] B. Michiels, I. Bogaert, J. Fostier, and D. De Zutter, "A Scalable, Parallel Algorithm for the Computation of the Translation Operator in the MLFMA, submitted to *IEEE Transactions on Antennas and Propagation*, 2013.